Software Developer Education - Applying Bloom Filters to Large, Real World Problems

R. Brad Tilley
Georgia Institute of Technology
Atlanta, Georgia
btilley@gatech.edu

ABSTRACT

In June 2017, The National Institute of Science and Technology (NIST) Special Publication 800-63-3b [12] established new guidelines with regard to how organizations should vet user passwords. Rather than composition policies that require a certain number of character sets, NIST now recommends that organizations check passwords against a list of banned passwords and reject those found on the list. As of July 2018, the 'Have I Been Pwned' list of known compromised passwords, alone, numbers more than half a billion strings [8] and this number is expected to grow larger as more online sites are compromised. This creates space and time efficiency challenges for software developers. A bloom filter [3], as described by Burton H. Bloom in his 1970 paper entitled, "Space/Time Trade-offs in Hash Coding with Allowable Errors", would be an efficient data structure to solve this problem. However, while software developers are very familiar with technologies and data structures such as relational databases, arrays, lists, trees and hash tables, they are not as familiar with less common, more abstract data structures. This work is intended to help software developers understand the complexity of lesser known data structures, such as bloom filters, and determine how and when to use them to solve large, real world problems.

ACM Classification Keywords

K.6.5. Security and Protection Unauthorized access (e.g., hacking, phreaking): Computing Milieux

Author Keywords

Cyber Security; Password Vetting; Software Developer Education.

INTRODUCTION AND PROBLEM BACKGROUND

Historically, organizations have implemented password policies to ensure the strength and resilience of user created passwords against guessing attacks. These policies are sometimes called "complexity policies" or "composition policies". They require end users to create passwords that are of some organizationally defined minimum length, and in addition to this, the password policies also require that user created passwords contain some number of uppercase, lowercase, digits and special characters. Alexander [1] wrote:

> When considering the characters used in a password it is useful to break the character set into groups: uppercase and lowercase letters each account for 26 characters, numbers account for only 10, and special characters account for 34. Strong passwords should have at least one character from each of three different groups.

Due to research such as Alexander's, many organizations implement a "three of four" character set requirement in password policies. This means that user generated passwords should be composed of characters from at least three of four defined character sets. For example, a typical "three of four" policy would require at least one uppercase character, one lowercase character and at least one digit or one special character.

The string, "password1" would not meet a "three of four" policy requirement as it only contains two of the four character sets (lowercase and digit), but the string "Password1" would meet the policy requirement as it contains three of the four sets (uppercase, lowercase and digit). And finally, the string "Password1!" would exceed the requirement due to the addition of a special character at the end.

Research shows that computer users have difficulty creating passwords that meet these composition policy requirements. In the paper, 'Password Education Based on Guidelines Tailored to Different Password Categories', by Kirsi Helkala [7], the author wrote:

> From the user perspective the general instructions [within password policies] are often too broad to be useful. One remembers best meaningful passwords, some like numbers and special characters while others recall best patterns from the keyboard, etc. The general instructions such as 'Minimum length of 8, use all character sets' do not support all users in their password generation process.

NIST 800-63-3b Appendix A [12] contains this statement about the effectiveness of password composition policies:

> As noted above, composition rules are commonly used in an attempt to increase the difficulty of guessing user-chosen passwords. Research has shown, however, that

CHI'16, May 07–12, 2016, San Jose, CA, USA

ISBN 123-4567-24-567/08/06...$15.00

DOI: http://dx.doi.org/10.475/123_4

> users respond in very predictable ways to the requirements imposed by composition rules [Policies]. For example, a user that might have chosen "password" as their password would be relatively likely to choose "Password1" if required to include an uppercase letter and a number, or "Password1!" if a symbol is also required.

Computer users often find ways in which to create passwords that meet the policy requirements yet are still easily guessed by attackers. Helkala [7] wrote:

> However, even if a system uses password policy, stating which character sets to use and what the minimum length of a password is, passwords might end up being predictable.

And further, Weir et al., [15] wrote:

> However, these policy mechanisms are hampered by an ill-defined understanding of their actual effectiveness against real attack techniques, and by circumvention strategies employed by the users. For example, a policy mandating that a user include at least three digits in a password will often result in the user simply appending "123" on the end of an insecure password.

In an effort to correct both the user issues with password creation policies as well as the ineffectiveness of these policies on actual password security and strength against password guessing attacks, NIST 800-63-3b [12] suggests that organizations stop implementing password composition polices and instead, allow users to create passwords however they like so long as the passwords are of a minimum length (at least eight characters), and that the passwords are not present in a list of banned passwords.

Specifically, NIST 800-63-3b [12] states:

> When processing requests to establish and change memorized secrets, verifiers SHALL compare the prospective secrets against a list that contains values known to be commonly-used, expected, or compromised.

And further:

> Verifiers SHOULD NOT impose other composition rules (e.g., requiring mixtures of different character types or prohibiting consecutively repeated characters) for memorized secrets.

These guidelines are a radical shift in how organizations accept and vet passwords created by users. They will create problems for all parties involved and increase solution complexity. End users will no longer be able to use their old tricks to satisfy the composition policies as those policies will no longer exist. Passwords that were once forbidden by corporate password composition policies, such as "jdhfyrgfbdhsas" before, would suddenly become acceptable under the new NIST guidelines. Software developers will have to devise ways to efficiently store and test a massive number (hundreds of millions or billions) of banned passwords rather than implement simple character set checks. Finally, corporate managers and IT compliance auditors will also have to modify corporate policies, internal procedures and guidelines.

The Need to Teach Complexity

A specific example is the need for computer and software technologists to learn and master more complex topics as the technical challenges faced by organizations grow and shift. As data sets grow larger and more distributed, it is common now for processors to have multiple cores and for programs to use multiple threads of execution. In addition, compute, storage and networking resources are quickly moving to the cloud.

In the paper, "Cloud computing - The business perspective", Marston et al. [11] wrote:

> [As] computing becomes more pervasive within the organization, the increasing complexity of managing the whole infrastructure of disparate information architectures and distributed data and software has made computing more expensive than ever before to an organization.

Further, the authors stated:

> Google's MapReduce or its opensource counterpart Hadoop makes the complex process of parallel execution of an application over hundreds of servers transparent to programmers.

Specifically with regard to larger data set complexity, Spafford [14] noted in his paper, "Preventing Weak Password Choices" that the list of banned passwords, used by the OPUS program contained 500,000 words. Today, Troy Hunt's research [8] shows that the list of compromised passwords, alone, is now more than 500 million and growing. On the surface, this problem may seem straight-forward. It is the same data set only larger. Why is there a need to think of the problem or its complexity differently?

Some new innovations, such as Google's MapReduce (mentioned above), hide much of the increased problem complexity. These new innovations show that the complexity we may need to learn about has more to do with the solution to the problem (in this case MapReduce), rather than the problem itself (efficiently process a larger data set over multiple computing resources).

For software developers, solutions to problems are constrained by physical resources. There is a need to be able to store, lookup and retrieve data using the space available in a reasonable period of time. Otherwise, the solutions will not be very practical or useful. In the paper, "Technology Diffusion and Organizational Learning", Attewell [2] noted that organizations depend upon technologists to figure out solutions to new and different problems. So while the fundamental problem may be the same as before (only larger or distributed) the thought process about the complexity of potential solutions may have changed significantly, and Attewell's research shows that organizations look to technologists to effectively and efficiently solve these problems.

To compound the challenges introduced by increasing scale and complexity, it is also generally difficult for humans to reason about abstract things. Things never experienced firsthand or used before can seem very foreign and at times almost unapproachable. Attewell [2] wrote:

> Of particular relevance are their observations that diffusing or deploying a technology is more difficult if (1) its scientific base is abstract or complex.

And, in some cases there is little to no contextual framework at all in the real-world that can be utilized to better understand complex or abstract topics. In the paper, "The Future of Engineering Education II. Teaching Methods that Work", Felder et al., [5] wrote:

> Cognitive research tells us that people learn new material contextually, fitting it into existing cognitive structures, 13-15 and new information that cannot be linked to existing knowledge is not likely to be retained.

In short, there is a need to find better ways to teach technologists how to think about more complex topics so that they can better solve larger, more abstract and distributed problems with newer, more complex solutions.

What Does Past Academic Research Say About Teaching Complex Topics?

Some approaches to effectively teaching abstract and/or complex topics seem promising. Research in several fields suggests that one approach to effectively teaching complex topics is to demonstrate or simulate the topics using concrete, real-world examples to provide context to students. Also, analogy (when real-world examples are not possible) has been identified as a successful method to teach abstract topics in some fields. Below are a few examples from academic research showing how demonstration, simulation and analogy may be reasonable approaches to teaching complex topics.

Demonstration

In the field of chemical engineering education research, Felder et al. [5] wrote:

> When presenting a new concept, start with a physical demonstration or real-world example, model the results, test the model through active experimentation and explore its implications. You might also find it worthwhile to have students measure their own learning styles and talk about the implications.

In the field of learning disabilities, Rivera and Smith's [13] paper, "Using a Demonstration Strategy to Teach Midschool Students with Learning Disabilities How to Compute Long Division", showed how demonstration could be used to effectively teach a complex topic, such as long division, to students with learning disabilities. The paper stated:

> Demonstration is documented as an effective teaching strategy for computational arithmetic. Smith (1973) and Smith and Lovitt (1975) studied the effectiveness of using demonstration when students with learning disabilities were learning computational skills... In this strategy, the teacher demonstrated the steps that led to the correct solution, and the solved problem remained available for reference while the student computed similar problems independently.

Simulation

With regard to simulation, Felder et al. [5] wrote:

> Provide visual illustrations and demonstrations of course-related material as possible. Most students get a great deal more out of visual information than verbal information... Show pictures, sketches, schematics, plots and flow charts, and computer simulations of process equipment and systems.

In the field of biology, Jordan et al. [9] used a computer program to simulate an aquarium to teach middle school students how to reason about complex biological ecosystems. In their paper, "Fostering Reasoning About Complex Systems: Using the Aquarium to Teach Systems Thinking", the authors wrote:

> Middle school learners find it difficult to understand ecosystems. Here we describe an intervention that pairs structure, behavior, and function (SBF) conceptual reasoning with computer-based learning tools that focus on an aquarium as a complex biological system. Based on results from 138 middle school students, we suggest that the use of SBF ontology combined with guided questions and simulations can enable students to consider multiple aspects of system dynamics.

Analogy

To teach the complex topic of quantum physics to young children, Hancock and Onsman [6] used analogy. In the paper, "Using analogy to teach complex concepts in science: The true story of 'Ellie the Electron'", they stated that:

> Scientific concepts are consistently described as challenging to learn and difficult to teach.

They also noted that analogy relies on an individual's existing knowledge to understand new complex topics. Thus, they created a character named 'Ellie' who lived in a world that was ruled by quantum physics. They wrote:

> The child's engagement is maintained through the characterization in the book. Ellie appears to be a regular little girl, with a personality - she gets up to mischief, however her world is different to ours and runs on different rules - that of quantum theory. Uncertainty is a defining feature of where she lives, which is another of Quantum Physics' essential truths.

Also, they noted that in this situation, analogy was the only method that could be used to effectively teach the topic due to a lack of any existing real-world reference or context:

> One of most conceptually difficult areas in science, quantum physics, is defined through statistical probabilities. Because we have no direct experience of the quantum world there is no other option but to describe it through analogy and metaphor. Generally, the verification of analogy depends entirely upon confirmation. In quantum physics no such verification is possible and therefore learning depends solely upon abstracted thought rather than direct observation.

There is a real need to more effectively teach complex and/or abstract topics to students. As shown above, this need impacts many fields of study and crosses many disciplines as well as all age ranges. Sometimes, both the problems that organiza-

tions face and their solutions are complex or abstract. And in general, organizations depend upon technologists to deal with increasing complexity so that they can progress and grow efficiently. By using demonstration, simulation and/or analogy (when required), educators may be able to more effectively teach complex and abstract topics to students.

PAST WORK RELATED TO BLOOM FILTERS

Bloom filters are abstract, probabilistic data structures that use bits to represent elements such as strings or numbers. They are space and time efficient and primarily intended to test for membership. A bloom filter hashes elements K times using a non-cryptographic hash function such as murmurhash and then stores K bits to identify the elements later during membership testing. Bloom filters have a constant lookup and insertion time of O(K) regardless of the number of elements represented in the filter. While tests for membership may never be falsely negative, they may be falsely positive. The rate of false positives can be controlled at the time of creation.

By using a bloom filter, rather than a list, database table or other more common data structure, software developers could efficiently store and test a massive number of compromised passwords each time a user creates or changes an account password. Bloom filters are very simple, however, due to the high abstraction level of bloom filters (elements are replaced and represented as bits), they are a lesser understood and thus lesser used data structure.

OPUS [14] was a tool proposed by Eugene H. Spafford, Department of Computer Sciences, Purdue University. It used a bloom filter to store and lookup commonly used passwords in a space and time efficient manner. At the time OPUS was proposed, the set of known bad passwords was much smaller than it is today. Spafford wrote:

> A UNIX version of OPUS is being constructed. It will be preloaded with a locally developed dictionary of almost 500,000 strings.

However, today, the set of known bad passwords from one online compromise can number in the tens of millions alone. For example, the Rockyou password dump [4] contained more than 15 million passwords. And, an aggregation of multiple password dumps can number in the hundreds of millions. The "Have I Been Pwned" password collection [8] contains more than 500 million exposed passwords as of July 2018.

This work contributes to the field by demonstrating that even though the list of bad passwords has grown exponentially, since OPUS [14] was proposed in 1992, that bloom filters are still an efficient and effective, if not often used, approach to solving this problem. Also, since NIST 800-63-3b [12] is a relatively new guideline (June 2017), more and more software developers will probably be faced with how to efficiently deal with this problem in the near future.

PROJECT GOALS AND DESIGN

The project developed instructional content using the Jupyter Notebook platform. The content is intended to inform and educate software developers by explaining and demonstrating a bloom filter as a space and time efficient solution to the National Institute of Science and Technology (NIST) 800-63-3b banned password checking requirement which requires organizations to test a massive list of banned passwords each time a user creates or changes a password. The project has three main goals:

1. Clearly explain what bloom filters are and why they were invented.

2. Help software developers understand and determine when to use bloom filters.

3. Demonstrate how bloom filters actually work by solving a real-world problem (The NIST 800-63-3b compromised password check).

The Jupyter Notebook that was developed for this project interleaves explanatory text, pictures and Python3 source code to better explain the abstract concepts behind bloom filters. According to T. Kluyver et al. [10], the Jupyter Notebook platform has been used to teach many complex scientific and computing topics. The authors wrote:

> We present Jupyter notebooks, a document format for publishing code, results and explanations in a form that is both readable and executable.

And further:

> Notebooks record a computation in order to explain it in detail to others, and a variety of tools help users to conveniently share notebooks.

Based on this, the Jupyter Notebook platform should be an ideal platform to educate software developers on how to apply a bloom filter to solve the password checking problem introduced by NIST 800-63-3b.

PROJECT FEATURES

Below are some screen captures, with explanatory text, from the Jupyter Notebook that was created. These are just a few brief highlights of the notebook content and features that are intended to reinforce key concepts underlying bloom filters.

Larry	Curly	Moe	Shemp
345	98	**57**	**587**
19	**43**	236	**43**
587	118	8	**57**

Figure 1. An example from the project Jupyter Notebook showing how a false positive may occur in a bloom filter. The three strings, 'Larry', 'Curly' and 'Moe' were added to the bloom filter, but the string 'Shemp' was not, however, the first three strings set all the bits that the string 'Shemp' would have set had it been inserted. This results in a false positive during membership testing.

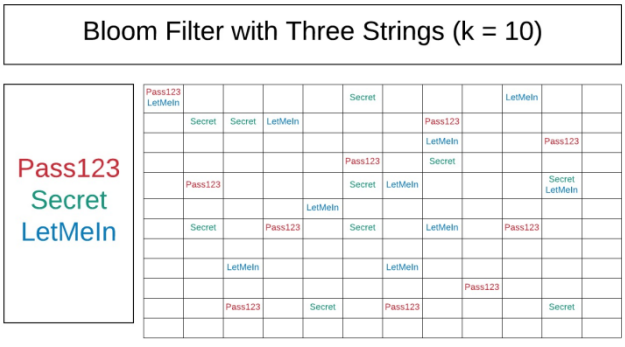


Figure 2. In this image taken from the project Jupyter Notebook, a grid graphic helps students visualize what a bloom filter looks like. Each grid square represents a bit in the bloom filter. Students can see from this picture how some elements inserted into the bloom filter may set the exact same bits. This visualization clarifies why it is impossible to remove elements from a traditional bloom filter as described by Burton H. Bloom.

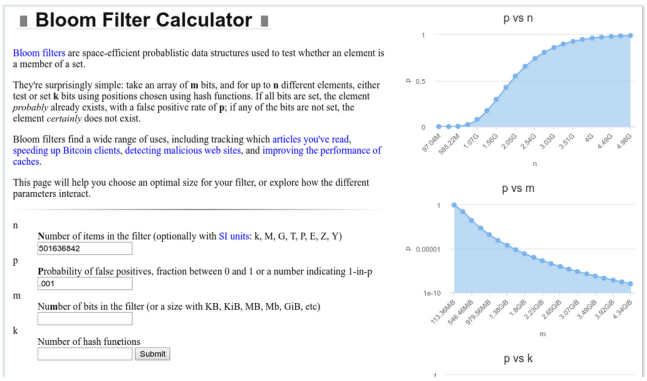


Figure 3. Here is a screenshot taken from an online Bloom Filter calculator written by Thomas Hurst https://hur.st/bloomfilter/. This screenshot is incorporated into the notebook to demonstrate to students how to size and design a bloom filter for optimal performance.

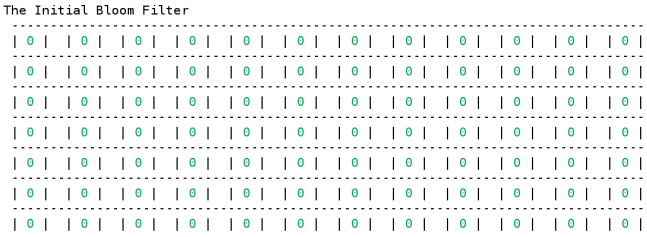


Figure 4. In this image taken from the Jupyter Notebook, students can visualize the state of an initial bloom filter before any elements have been inserted. This picture is actually generated from the Python3 code in the notebook interactively. Each time another element is added to the bloom filter, the state of the filter is updated (along with this picture) to reflect the new bits that were set.

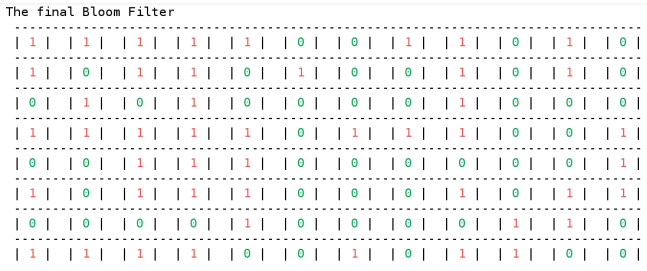
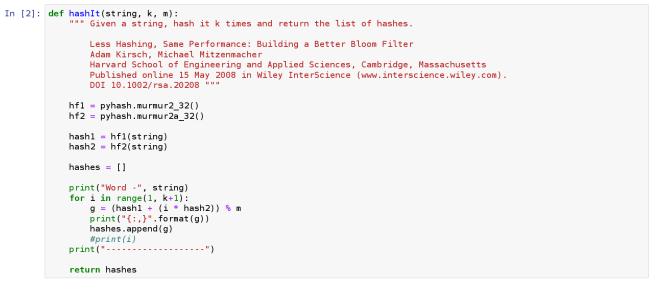


Figure 5. In this image taken from the Jupyter Notebook, students can visualize the state of the final bloom filter after all elements have been inserted. If the bloom filter has been properly sized, roughly half the bits should be set st this point in time.



```
In [2]: def hashIt(string, k, m):
            """ Given a string, hash it k times and return the list of hashes.

                Less Hashing, Same Performance: Building a Better Bloom Filter
                Adam Kirsch, Michael Mitzenmacher
                Harvard School of Engineering and Applied Sciences, Cambridge, Massachusetts
                Published online 15 May 2008 in Wiley InterScience (www.interscience.wiley.com).
                DOI 10.1002/rsa.20208 """

            hf1 = pyhash.murmur2_32()
            hf2 = pyhash.murmur2a_32()

            hash1 = hf1(string)
            hash2 = hf2(string)

            hashes = []

            print("Word -", string)
            for i in range(1, k+1):
                g = (hash1 + (i * hash2)) % m
                print("{:,}".format(g))
                hashes.append(g)
                #print(i)
            print("-------------------")

            return hashes
```

Figure 6. In this image taken from the Jupyter Notebook, students can interactively execute code to build and test a bloom filter for membership as they step through the notebook content. This code builds and tests a very large bloom filter (more than 500 million items) to solve the NIST 800-63-3b password checking guideline discussed earlier.

FUTURE DEVELOPMENT

As stated earlier in this paper, in addition to software developers, the radical changes and increased solution complexity introduced by NIST 800-63-3b password vetting recommendations will also impact other groups across various fields and industries. End computer users will need to learn how to think differently about password creation as their old tricks to satisfy corporate password composition policies will no longer work. Corporate IT managers and IT security/compliance auditors (who define policies and verify compliance to them) will need to learn how to let go of the old methods and to embrace the new ones. These latter groups will need to be convinced that the new guidelines are indeed more secure and safer than what they have grown accustomed to and comfortable with over the years. Solutions to these challenges may be a good area of future work for other educational technology researchers.

CONCLUSION

The project developed content using the Jupyter Notebook platform to teach software developers what bloom filters are and how and when to use them. The content combined instructional text, graphics along with executable source code. Demonstration was the pedagogical method used to address the complexity of teaching bloom filters. Finally, the problem selected for the bloom filter demonstration was based on a recent, large problem created by NIST 800-63-3b to help students experience how bloom filters can be applied to solve concrete, real world problems in a space and time efficient manner.

ACKNOWLEDGMENTS

The author would like to acknowledge and thank Ken Brooks and David Joyner for guidance and recommendations during the project.

REFERENCES

1. Steven Alexander. 2004. password protection for modern operating systems. *;login:* 29, 3 (2004). Retrieved July 6, 2018 from **https://www.usenix.org/system/files/login/articles/1103-alexander.pdf**.
2. Paul Attewell. 1992. Technology Diffusion and Organizational Learning: The Case of Business Computing. *Organization Science* 3, 1 (1992), 1–19. DOI: **http://dx.doi.org/10.1287/orsc.3.1.1**
3. Burton H. Bloom. 1970. Space/Time Trade-offs in Hash Coding with Allowable Errors. *Commun. ACM* 13, 7 (July 1970), 422–426. DOI: **http://dx.doi.org/10.1145/362686.362692**
4. Ron Bowes. 2018. The Rockyou Exposed Password List. Website. (15 July 2018). Retrieved July 15, 2018 from **https://wiki.skullsecurity.org/Passwords**.
5. Richard M Felder, Donald R Woods, James E Stice, and Armando Rugarcia. 2000. The future of engineering education II. Teaching methods that work. *Chemical Engineering Education* 34, 1 (2000), 26–39.
6. Yvette Hancock and Andrys Onsman. 2005. Using analogy to teach complex concepts in science: The true story of 'Ellie the Electron. (07 2005). Retrieved July 7, 2018 from **https://www.aare.edu.au/data/publications/2005/han05274.pdf**.
7. Kirsi Helkala. 2011. Password Education Based on Guidelines Tailored to Different Password Categories. *JOURNAL OF COMPUTERS* 6 (05 2011), 969–975.
8. Troy Hunt. 2018. The Have I Been Pwned Exposed Password List. Website. (15 July 2018). Retrieved July 15, 2018 from **https://haveibeenpwned.com/Passwords**.
9. Rebecca C. Jordan, Cindy Hmelo-Silver, Lei Liu, and Steven A. Gray. 2013. Fostering Reasoning About Complex Systems: Using the Aquarium to Teach Systems Thinking. *Applied Environmental Education & Communication* 12, 1 (2013), 55–64. DOI: **http://dx.doi.org/10.1080/1533015X.2013.797860**
10. Thomas Kluyver, Benjamin Ragan-Kelley, Fernando PÃl'rez, Brian Granger, Matthias Bussonnier, Jonathan Frederic, Kyle Kelley, Jessica Hamrick, Jason Grout, Sylvain Corlay, Paul Ivanov, DamiÃąn Avila, Safia Abdalla, Carol Willing, and Jupyter development team [Unknown. 2016. Jupyter Notebooks - a publishing format for reproducible computational workflows. (01 2016). Retrieved July 7, 2018 from **https://pdfs.semanticscholar.org/e478/68841d87efe261451a43b00d6c81cf7fb7a3.pdf**.
11. Sean Marston, Zhi Li, Subhajyoti Bandyopadhyay, Juheng Zhang, and Anand Ghalsasi. 2011. Cloud computing - The business perspective. *Decision support systems* 51, 1 (2011), 176–189.
12. National Institute of Standards and Technology. 2017. NIST Special Publication 800-63B Digital Identity Guidelines Authentication and Lifecycle Management. Website. (30 June 2017). Retrieved July 15, 2018 from **https://pages.nist.gov/800-63-3/sp800-63b.html**.
13. Diane Rivera and Deborah Deutsch Smith. 1988. Using a Demonstration Strategy to Teach Midschool Students with Learning Disabilities How to Compute Long Division. *Journal of Learning Disabilities* 21, 2 (1988), 77–81. DOI: **http://dx.doi.org/10.1177/002221948802100203** PMID: 3346610.

14. Eugene H. Spafford. 1992. OPUS: Preventing Weak Password Choices. *Computers & Security* 11 (1992), 273–278.

15. Matt Weir, Sudhir Aggarwal, Michael Collins, and Henry Stern. 2010. Testing Metrics for Password Creation Policies by Attacking Large Sets of Revealed Passwords. In *Proceedings of the 17th ACM Conference on Computer and Communications Security (CCS '10)*. ACM, New York, NY, USA, 162–175. DOI: http://dx.doi.org/10.1145/1866307.1866327